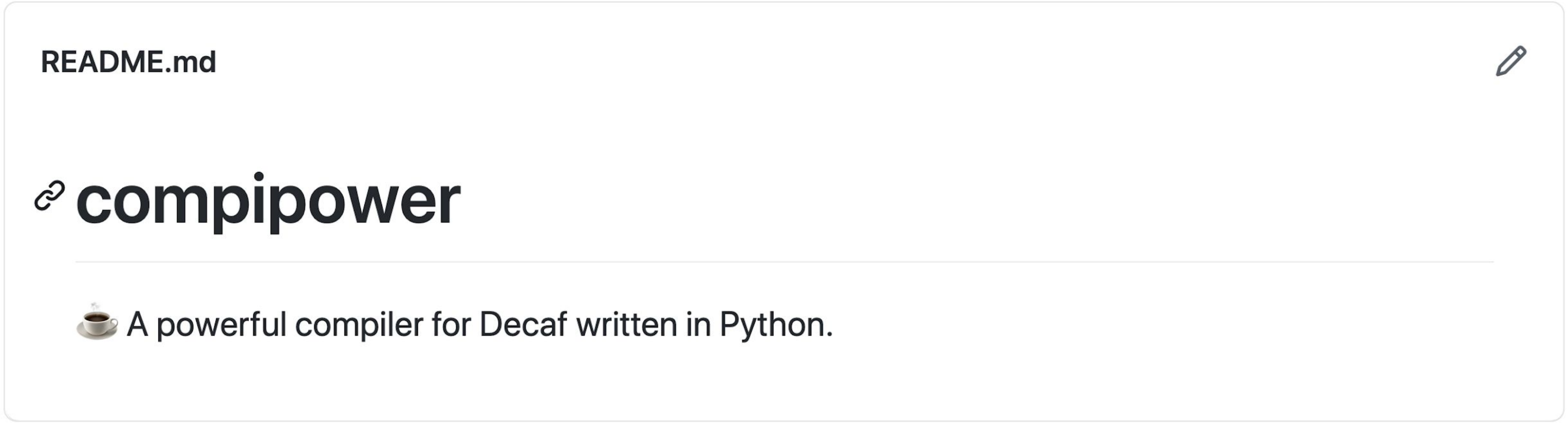
Design docs



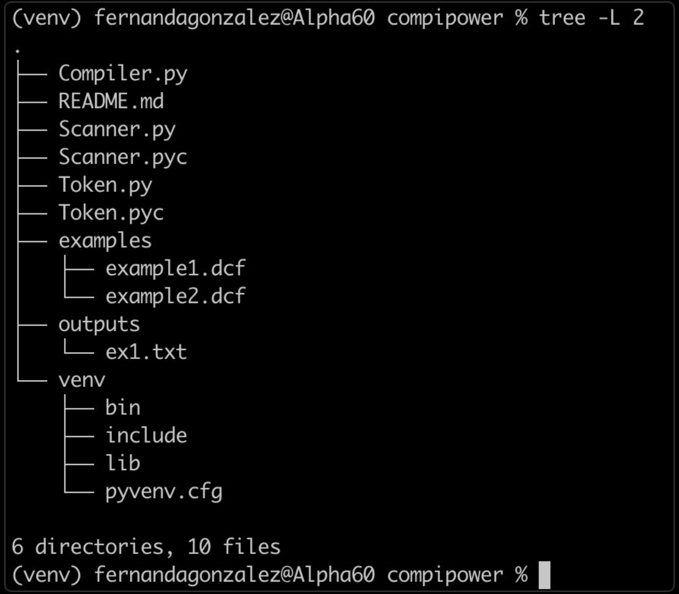
README.md

compipower

☕ A powerful compiler for Decaf written in Python.

Abner Xocop Chacach
abnerxch

Fer González
armi3



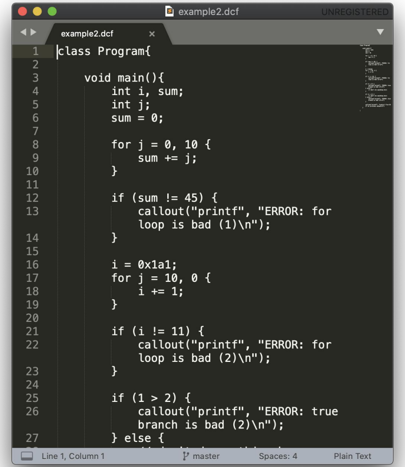
```
(venv) fernandagonzalez@Alpha60 compipower % tree -L 2
.
├── Compiler.py
├── README.md
├── Scanner.py
├── Scanner.pyc
├── Token.py
├── Token.pyc
├── examples
│   ├── example1.dcf
│   └── example2.dcf
├── outputs
│   └── ex1.txt
└── venv
    ├── bin
    ├── include
    ├── lib
    └── pyvenv.cfg

6 directories, 10 files
(venv) fernandagonzalez@Alpha60 compipower %
```

Compiler structure

Proyecto disponible en Github.

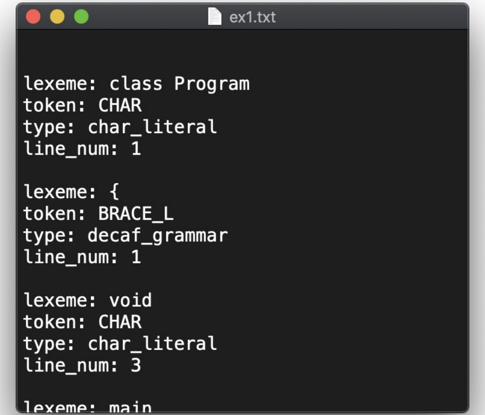
Compiler structure

Input: program



```
class Program{

    void main(){
        int i, sum;
        int j;
        sum = 0;

        for j = 0, 10 {
            sum += j;
        }

        if (sum != 45) {
            callout("printf", "ERROR: for
            loop is bad (1)\n");
        }

        i = 0x1a1;
        for j = 10, 0 {
            i += 1;
        }

        if (i != 11) {
            callout("printf", "ERROR: for
            loop is bad (2)\n");
        }

        if (1 > 2) {
            callout("printf", "ERROR: true
            branch is bad (2)\n");
        } else {
```

Output: token stream



```
lexeme: class Program
token: CHAR
type: char_literal
line_num: 1

lexeme: {
token: BRACE_L
type: decaf_grammar
line_num: 1

lexeme: void
token: CHAR
type: char_literal
line_num: 3

lexeme: main
```

Parser

Scanner

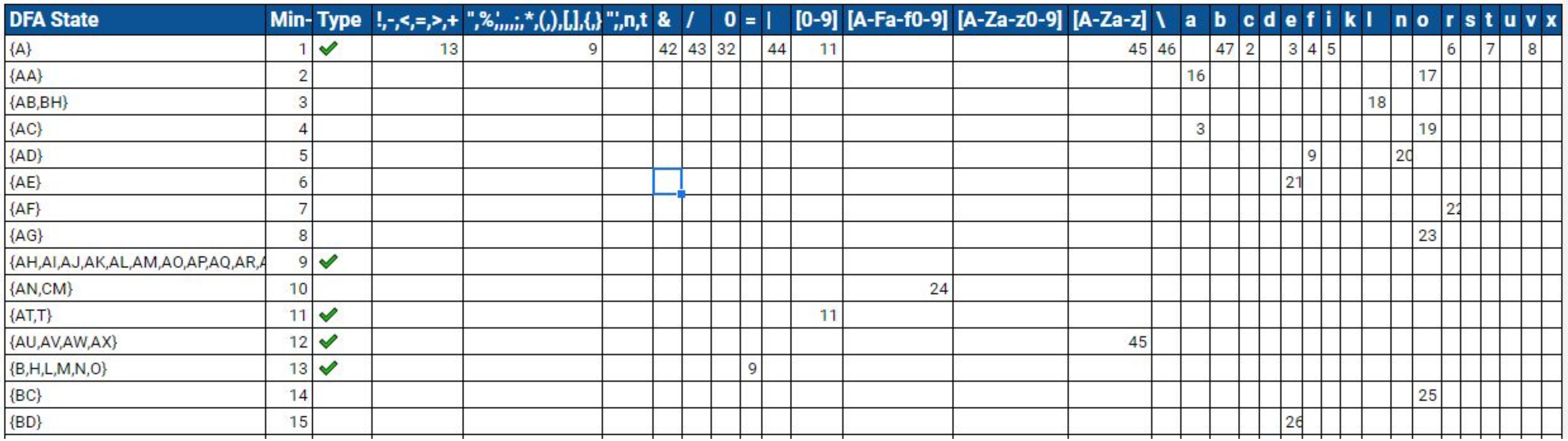
$$\begin{tikzcd}\text{Decaf grammar} \arrow[r] & \text{NFA} \arrow[r] & \text{DFA} \arrow[r] & \text{Regular expression}\end{tikzcd}$$

Scanner

NFA disponible en

[shorturl.at/sNUZ4]

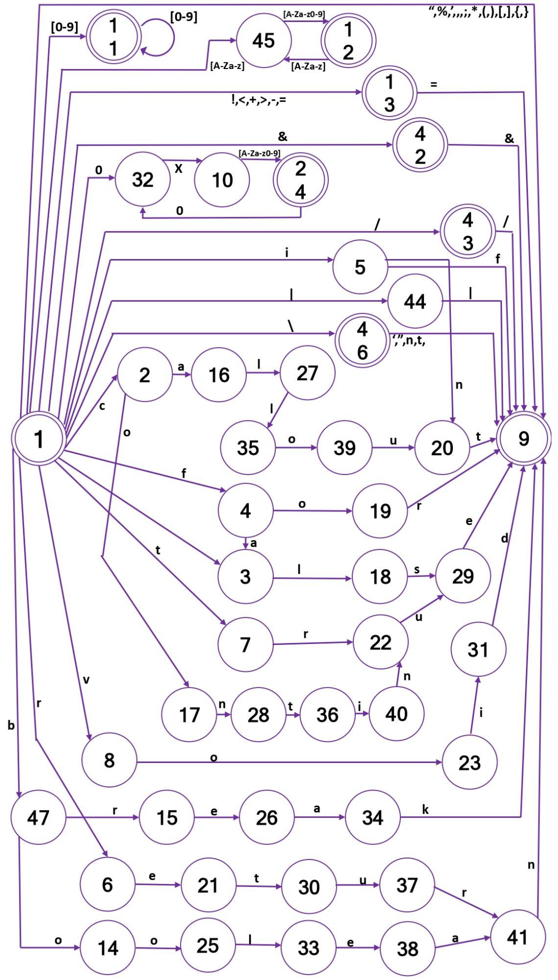
NFA



DFA State	Min-	Type	!,-,<,=,>,+	",%,',,,,,:,;,*,(,),[,],{,}	"',n,t	&	/	0	=	|	[0-9]	[A-Fa-f0-9]	[A-Za-z0-9]	[A-Za-z]	\	a	b	c	d	e	f	i	k	l	n	o	r	s	t	u	v	x
{A}	1	✓	13	9		42	43	32		44	11			45	46		47	2		3	4	5					6		7		8	
{AA}	2															16										17						
{AB,BH}	3																							18								
{AC}	4															3										19						
{AD}	5																				9				20							
{AE}	6																			21												
{AF}	7																										22					
{AG}	8																									23						
{AH,AI,AJ,AK,AL,AM,AO,AP,AQ,AR,A	9	✓																														
{AN,CM}	10											24																				
{AT,T}	11	✓									11																					
{AU,AV,AW,AX}	12	✓												45																		
{B,H,L,M,N,O}	13	✓							9																							
{BC}	14																									25						
{BD}	15																			26												

DFA table completo en https://bit.ly/33TFKHi

DFA



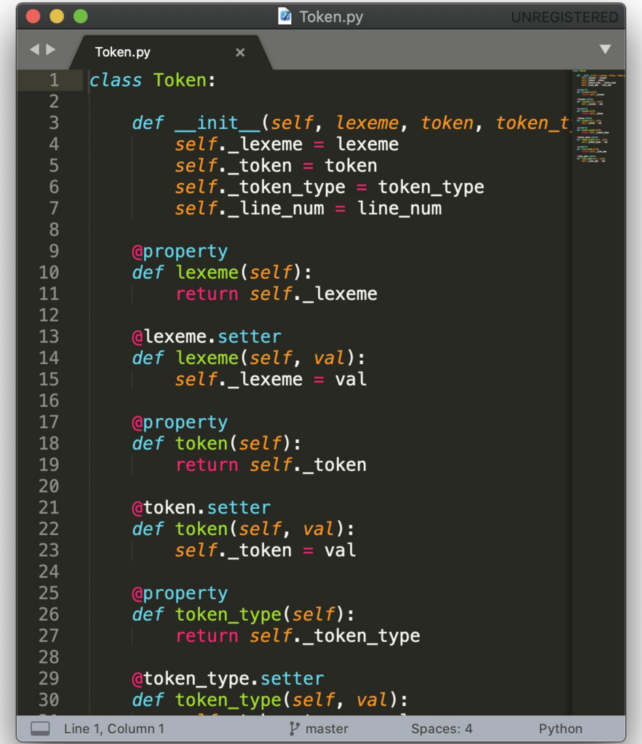
DFA disponible en

https://bit.ly/3ctaUZP

DFA

```
class Program|
{|}|\[|\]|,|;|=|\-|\+|\-=|\!|<|>|<=|>=|==|\!=|\+
=|\*|\/|\&\&|\|\||%|\/\/|\"|\'|\\|\'|\"|\n|\t|\)
|\(|int|boolean|if|for|return|break|continue|cal
lout|true|false|void|else|0[x|X][\w|\d]+|\d+|[\w
][\w\d_]+|[\w]+
```

Regular expression



```python
class Token:

    def __init__(self, lexeme, token, token_t
        self._lexeme = lexeme
        self._token = token
        self._token_type = token_type
        self._line_num = line_num

    @property
    def lexeme(self):
        return self._lexeme

    @lexeme.setter
    def lexeme(self, val):
        self._lexeme = val

    @property
    def token(self):
        return self._token

    @token.setter
    def token(self, val):
        self._token = val

    @property
    def token_type(self):
        return self._token_type

    @token_type.setter
    def token_type(self, val):
```

Anatomy of a token

run DEMO